1983

Utilizing attached microprocessors under UNIX*

Hsiao-Hwa Shirley Ko

Rochester Institute of Technology
School of Computer Science and Technology

UTILIZING ATTACHED MICROPROCESSORS UNDER UNIX*

by

HSIAO-HWA (SHIRLEY) KO

A thesis submitted to
The Faculty of the School of Computer Science and Technology
in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science

December, 1983

Approved by:

Peter H. Lutz 2/1/84
Thesis Advisor Dr. Peter H. Lutz Date

Rayno Niemi 2/1/84
Committee Member Dr. Rayno D. Niemi Date

Kenneth A. Reek 2/1/84
Committee Member Mr. Kenneth A. Reek Date

* UNIX is a trademark of Bell Laboratories

Title of Thesis: UTILIZING ATTACHED MICROPROCESSES UNDER UNIX*

I, Shirley Hsiao-Hwa Ko prefer to be contacted each time a request for reproduction is made. I can be reached at the following address:

Date: Jan 26, 1984

Acknowledgement

I would like to thank my thesis advisor, Dr. Peter Lutz, for spending so much time directing my efforts on this project. I would also like to thank Mr. Kenneth Reek for the ideas for this project and for the time he spent in helping me. I would also like to thank Dr. Rayno Niemi for the time he has spent as a member of my committee.

Finally, I would like to thank my husband, Ya-Tien, for his encouragement and patience.

TABLE OF CONTENTS

1. Introduction and Background

1.1. Introduction

Computer systems have rapidly changed from single computer serving all of the organization's computational needs to a large number of separate but interconnected computers to do the job. The merging of computers and communications has profoundly influenced the way computer systems are organized. The development of computer networks has enabled computers to exchange information and to talk to each other. Also, the load of the c ntral system can be shared by its attached satellite systems. The system to be improved here contains two different computers: the PDP-11/34 and the LSI-11. The LSI-11 is a 16-bit microcomputer which, at R.I.T., is connected to a PDP-11/34 as a peripheral device. It has limited I/O capability and no operating system. The role it can play in the whole system is therefore limited.

Currently, there are three LSI-11 microcomputers attached to the PDP-11/34. The objective of this thesis is to develop a satellite distributed computing network to be used among these four machines. The purpose of implementing a distributed computing network is to obtain the advantage of saving the PDP-11/34 CPU time, and increasing the utility of the LSI-11.

The main body of this thesis are two programs which have been implemented on the PDP-11/34 and LSI-11 separately. They cooperate to make the whole system work as a single distributed system. The principles and procedures of these two programs will be described in the following chapters.

1.2. Background

It is well known that the dramatic developments of the last few years in integrated circuit technology have revolutionized computer hardware. Also, great progress in communication technology is being made by providing substantially cheaper interfaces and simpler protocols. The conjunction of these two developments has led to a particular style of distributed computing in which the work of a system is performed by a collection of computers connected via a network. The goal of this network is, if possible, to obtain high performance through the use of many low-cost processor elements under the management of an effective operating system. Many such network systems are under development. Each of them can be used for particular purposes and placed where appropriate for that purpose. Four network operating systems will be discussed here to compare their advantages. Since they are all implemented on PDP-11 family systems, which are similar to the one at R.I.T., their approaches to running a distributed system are directly relevant to this

project.

(1) The Carnegie-Mellon multi-mini-processor, c.mmp.: It is a multi-processor network system constructed at Carnegie-Mellon University [30, 31]. It consists of 16 processors, all of them are various models of PDP-11 mini-computers, together with 16 memory modules and a 16 * 16 crosspoint switch (see Figure 1-1). Each processor is actually an independent computer system with a small amount of private memory and various I/O devices. The kernel of an operating system for c.mmp, called HYDRA, builds a set of mechanisms from which an arbitrary set of operating system facilities and policies can be conveniently, flexibly, efficiently and reliably constructed.

(2) The Cm* system: Another new multiprocessor system (see Figure 1-2), developed at the same university, consists of multiple DEC LSI-11 microprocessors which are grouped in clusters [20]. Each LSI-11 in a cluster, on which hang some local memory and I/O devices, is connected through a switch to a map bus. Each map bus has a centralized bus arbiter, the Kmap, that mediates intra- and inter- cluster data transfers. The advantage of the Kmap is that it provides access to the memory of the other processors by communicating with them via an inter-cluster bus.

(3) The Cambridge Model Distributed System (CMDS): The CMDS is a network operating system used to support the Cambridge Digital Communications Ring (shown in Figure 1-3) within the computer laboratory at the University of Cambridge [19, 29]. The network consists of a number of interconnected microcomputers, called " servers " , and minicomputers. The servers implement a specific operating system function and provide several basic services for all users. The minicomputers, which function as computational machines, can be allocated to a user upon request. Communication is done by sending messages along the ring, with each module checking all message headers for messages sent to it. This system provides interactive time sharing facilities to its users via the network.

The advantage of the server approach is that it is flexible in selecting the server functions, e.g. editor server, without necessarily allocating a computational machine. This saves the time needed to get access to a computational machine. Another advantage is that if a bottleneck develops, extra servers can be added to the system to solve the problem.

(4) The satellite processor system (SPS): It is a software support system for a network of minicomputers and microcomputers which have evolved over the years at Bell Laboratories [16]. The fundamental concept involved in

supporting these satellite processors is the extension of the central processor operating system to each satellite processor. Software interfaces permit a program in the satellite processor to behave as if it were running in the central processor. This allows a user program, which might normally run in the central processor using its operating system, to run in an satellite processor with no resident operating system. The satellite processor concept permits the advantages of a large computing system to be extended to many attached miniprocessors, giving each satellite processor access to the central processor's file system, software tools, and peripherals while retaining the real-time response and flexibility of a dedicated minicomputer. One particular SPS hardware configuration includes a DEC PDP-11/45 central computer, several LSI-11 microcomputers, and a number of other satellite processors attached using a serial I/O loop as the communication links (see Figure 1-4). The implementation of this system was considerably simplified by the fact that all processors, central and satellite, belong to the same family of computers.

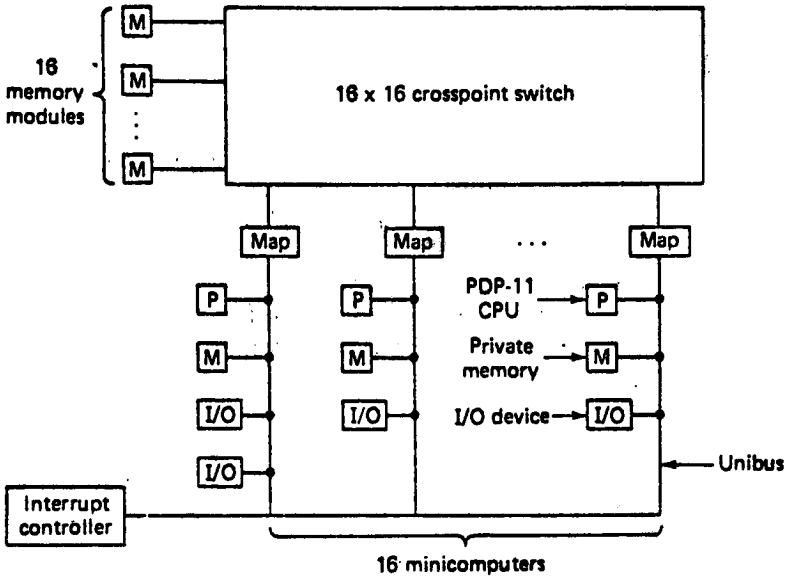


Figure 1-1 C.mmp.

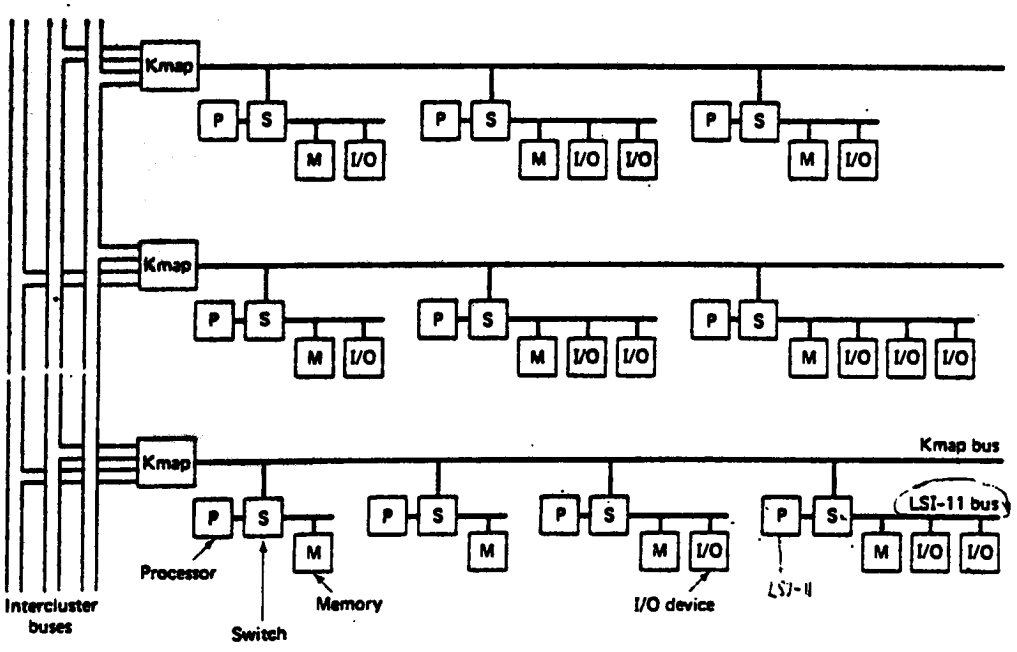


Figure 1-2. A three cluster Cm* System

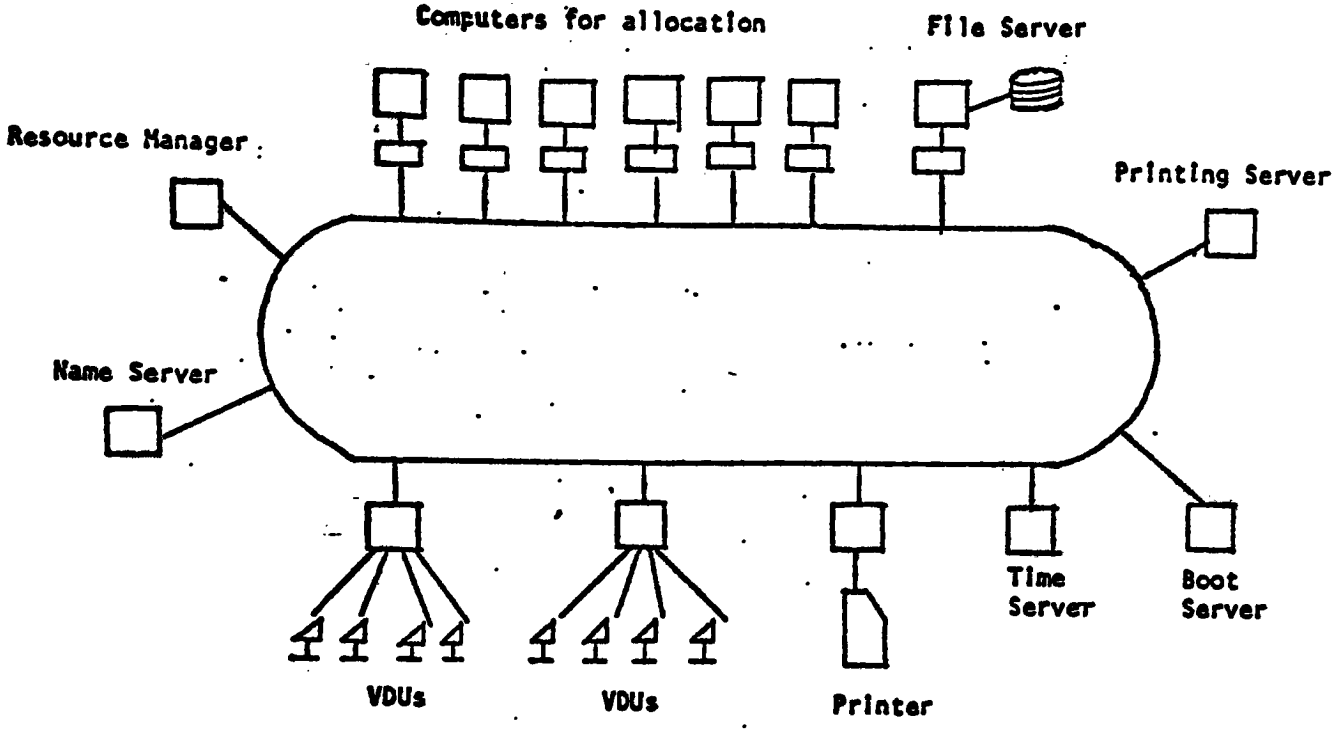


Figure 1-3. Cambridge Model Distributed Ring

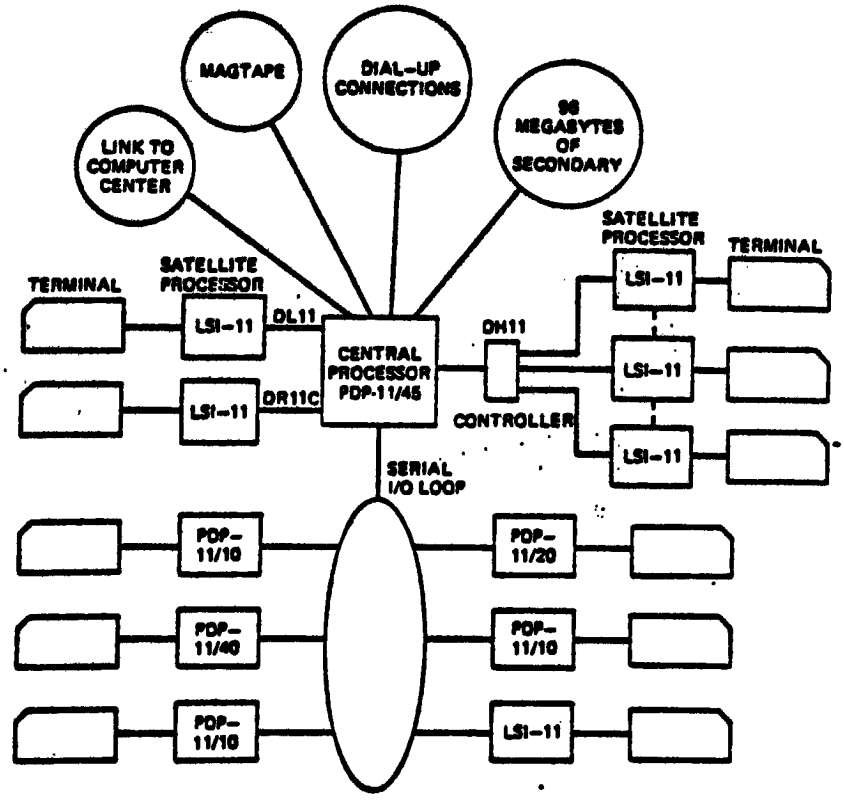


Figure 1-4. Satellite Processor System

After comparing the above four systems, one finds that the existing system at R.I.T. (see Figure 1-5) is most similar to the last one, i.e. SPS. Bearing in mind the concept of a satellite distributed system, we may begin to implement a prototypical distributed operating system.

The language " C " is used as the major programming language since it is the most efficient high level language under the UNIX operating system. Assembly language is also used whenever it is impossible to use C or when there are special efficiency concerns.

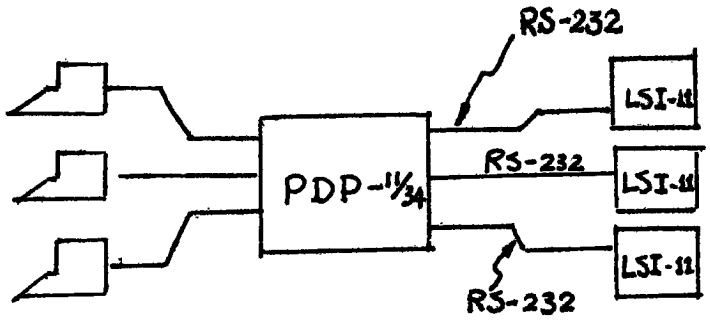


Figure 1-5. System Configuration in R.I.T.

2. System Overview

2.1. Introduction

This chapter provides a general idea of the system configuration. Major specifications are also briefly stated.

2.2. System Description

The LSI-11 microcomputer is attached to the PDP-11/34 minicomputer by the RS-232c interface. The user terminal is hooked up directly to the PDP-11/34. From the viewpoint of communications among these three units, there are two interfaces in this distributed system. Their connections are schematically shown in Figure 2-1.

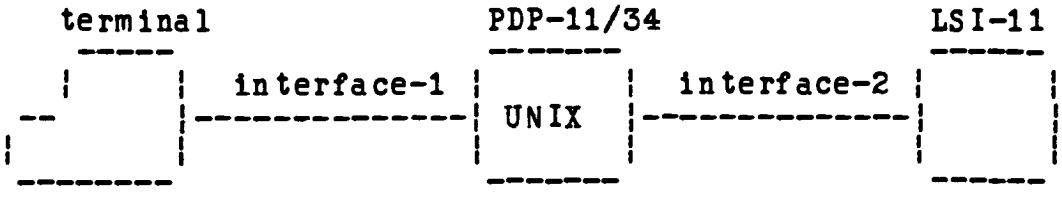


Figure 2-1.

In this thesis, two major programs are implemented. One is " LSIRUN ", which is executed on the PDP-11/34 under UNIX operating system. The other is " KERN ", which is loaded into the LSI-11 as a kernel program combined with the user program. The kernel program mimics the UNIX operating

system which runs on the PDP-11/34. Those system calls that can be accommodated by the LSI-11 are executed directly on the microprocessor under the supervision of the kernel program. Those that cannot are transmitted by KERN to LSIRUN, which then executes the system calls on the PDP-11/34 and passes the results back to KERN.

A programmer sitting at the terminal logs onto the PDP-11/34 and uses the UNIX editor to create or update a program source file which will run on the LSI-11. When the program is ready, the user program combines the kernel program and the library routines on UNIX, generating a KERN object file. When the user types the LSIRUN command followed by the KERN name, the KERN file is loaded into the LSI-11 and starts to execute the user program there. The user terminal is now the standard input and output device for the LSI-11. From this terminal, the user then observes the results of running the program. If any program changes are required, the preceding steps are repeated.

This design requires that the functions of the Octal Debugging Technique (ODT) be preserved to facilitate debugging. The user can debug his program by halting the LSI-11 manually, then having the system return to the ODT mode and debug his program. Console ODT executes as a portion of the processor microcode that allows the processor to respond to commands and information via the terminal. Since ODT resides in microcode it cannot be changed.

Communication between the user and processor is through a stream of ASCII characters interpreted by the processor as console commands.

Normally, terminal input is processed in units of lines (called normal mode). This means that the reading of a program through interface-1 in normal mode will be briefly suspended each time a new line is being entered, and will resume when the carriage return is hit. However, through interface-2, communication is character by character (cbreak mode). In order to send the ODT address value down to the LSI-11, LSIRUN on the PDP-11/34 must change the normal mode for interface-1 into cbreak mode. Input data can then be read character by character, as soon as it is typed, without the need for terminating new lines. This is necessary because ODT expects to respond to characters in this fashions. After executing the ODT mode, the interface-1 must revert back to normal mode for normal I/O transmission. LSIRUN on the PDP-11/34 must be capable of identifying different situations and changing interface-1 accordingly. The relationship of communications in the system is shown in Figure 2-2.

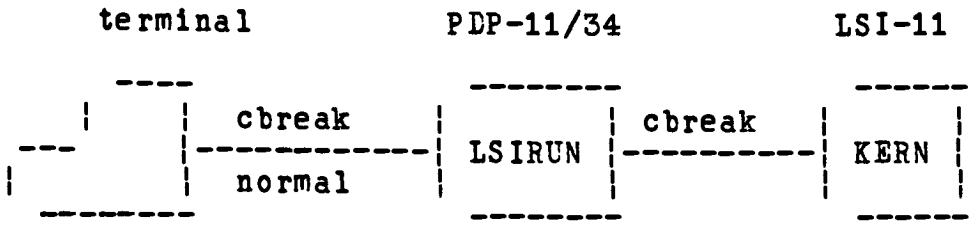


Figure 2-2

In order to accelerate the process of console ODT data, LSIRUN forks a child process called LSIBACK, which only listen to the LSI-11 during execution of the console ODT, and writes the output data to the current terminal screen. As soon as the user types the running command, LSIBACK must be killed and terminal interface-1 must be changed back to normal mode. The ODT environment is illustrated in Figure 2-3. The environment in which the user program is executed by communicating between the two CPUs are shown in Figure 2-4.

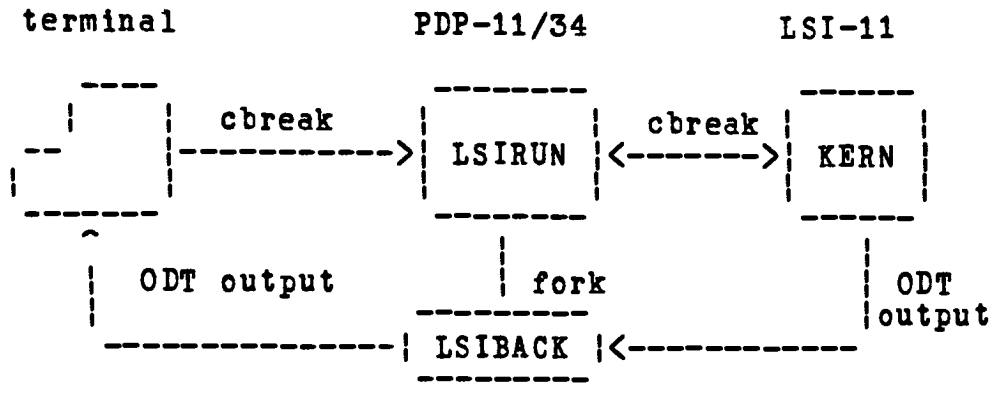


Figure 2-3.

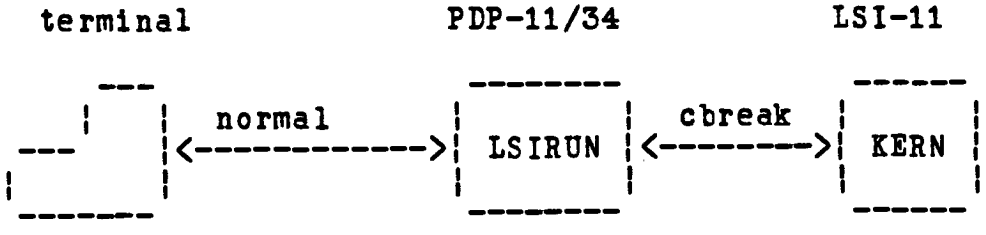


Figure 2-4.

3. System Implementation

3.1. Introduction

Developing this small satellite processor system is simplified by the fact that all processors involved , PDP-11/34 and LSI-11s, belong to the same family of computers (DEC PDP-11 series). The UNIX operating system on the PDP-11/34, which is a central computer, controls the loading, running and debugging in the attached LSI-11s. The fundamental concept involved in supporting these LSI-11s is the extension of the UNIX operating system to each LSI-11 microcomputer. This chapter discusses the system architecture for the PDP-11 families. Also, the software interface programs, LSIRUN and KERN, are described in detail.

3.2. System Architecture

The hardware configuration described in this chapter consists of a DEC PDP-11/34 computer with a number of attached LSI-11 microcomputers. All components of the PDP-11/34, i.e., a CPU, a main memory, and various I/O devices, are connected by a single bus called a UNIBUS (see Figure 3-1). It is used for communication among these components. On the other hand, the Q-bus is the connection among the components of LSI-11 and used for transmitting information from one component to another. Both busses are the single 16-bit, communication path in their system. The similarity

of the Q-bus to the UNIBUS enables the development of large amounts of expertise in the PDP-11 family interfacing.

The CPU features which are common throughout the PDP-11 family include:

- The PDP-11 Basic Instruction Set.
- Eight General Purpose Registers.
- Addressing Modes.
- ODT/ASCII Console Routine (option).
- A Real-Time Clock Interrupt (option).

Refer to the following bibliography for more general architecture information [7, 8, 9, 11, 23].

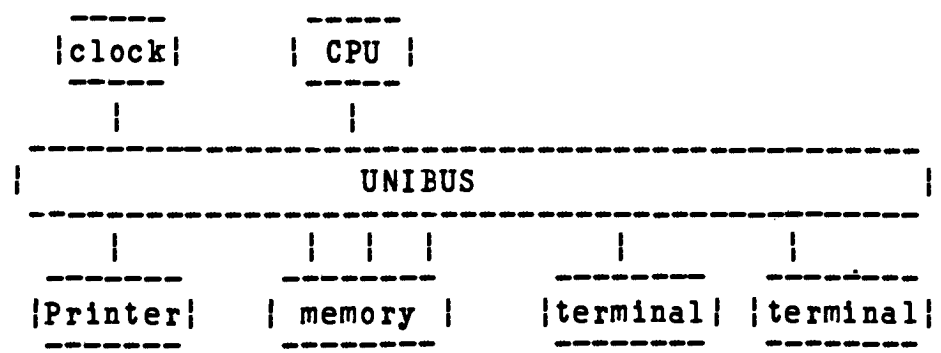


Figure 3-1. A PDP-11 System

3.3. Communication Links

The satellite processor concept extends the UNIX operating system on the PDP-11/34 to its attached microprocessor, the LSI-11. The interface between a user program running on the PDP-11/34 and UNIX is by means of system calls. Similarly, the interface between a user program running on the LSI-11, and the UNIX environment which is being emulated, is also by means of system calls (see Figure 3-2), except that here the extension is achieved by trapping system calls in the LSI-11. System calls related to the file system and I/C system are then passed with their arguments to the PDP-11/34. Other system calls, which can be accommodated on the LSI-11, are executed directly on the microprocessor.

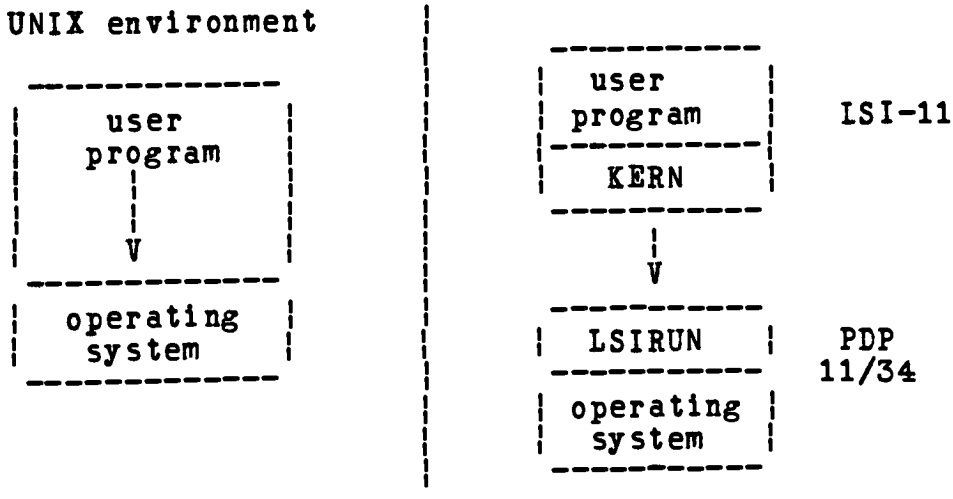


Figure 3-2.

The method of data transfer between the PDP-11/34 and the LSI-11 is a terminal interface. A program is required on the LSI-11 to deliver data to the PDP-11/34. This is done by loading the data, character by character, with the terminal printer buffer (TPB) in the LSI-11. The address of the (TPB) is prescribed by the hardware.

The CPU of LSI-11 can send data to the terminal printer buffer much faster than the receiver can respond to it. The printer buffer can hold only one character and, unless the current character is used, any attempt to send a new character to the buffer will fail. Consequently, if the CPU sends too rapid a stream of characters, many will be lost.

To ensure that each character gets sent, the CPU must check before sending it that the printer buffer is ready to receive it. This check is made by referring to the terminal printer status register, (TPS). The status register, like the printer buffer, has an address; the TPS is one word address below the TPB. These two locations in LSI-11 are:

```
TPS = 177564          ; terminal printer status
TPB = 177566          ; terminal printer buffer
```

Bit 7 of the TPS is the " ready " bit; this bit is clear when a character is being processed in the printer buffer and is set when the buffer is ready to receive a new character. Any attempt by a program to write into this bit is ignored; i.e., it is a read only bit.

Just as with the printer buffer, the terminal keyboard is wired to the computer in a way that allows addressing of its buffer and status register. These two locations are:

```
TKS = 177560          ; keyboard status
TKB = 177562          ; keyboard buffer
```

Thus the CPU must be coordinated with terminal input just as with output. As soon as a character is read from the TKB, the terminal clears the done bit in the TKS to ready it for the next input character.

Data is transmitted through the terminal print buffer and keyboard buffer from PDP-11/34 to LSI-11, and vice versa.

3.4. KERN Software

" KERN " is an object file which is the combination of a kernel program and an user program. The function of KERN includes initializing interrupt vector, handling traps and interrupts properly, and providing a communication environment so as to transmit information between the PDP-11/34 and the LSI-11.

3.4.1. The Algorithm of the KERN Program

The kernel program supports a single-user system with only one process running at any one time in the LSI-11. Most of the system calls that relate to I/O system or file system need to be implemented by communication with the PDP-11/34. A few system calls that can be accommodated by the LSI-11 are executed directly on the microprocessor. The algorithm of the kernel program is listed below:

1. Initialize the interrupt vectors for internal and external interrupts.

2. Jump to the user program.

3. When a system call occurs, trap to an interrupt handler.

   3-1. Check the system call. If it can be done on the LSI-11, do it directlyand retrun to the user program.

   3-2a. If it can not be implemented on the LSI-11, the arguments of the system call are sent to the PDP-11/34 and the LSI-11 awaits a reply.

   3-2b. Return to user program after the result from PDP-11/34 is ottained.

Some relations do not exist in this simple distributed system in comparison to those in UNIX system. Therefore,

those corresponding system calls are not implemented. The deleted system calls are:

(1). acct - turn accounting on or off

(2). getiword - fetch word from user instruction space

(3). mpx - create and manipulate multiplexed files

(4). mpxcall - multiplexed and channel interface

(5). nice - set program priority

(6). phys - allow a process to access physical

(7). profil - execution time profile

(8). ptrace - process trace

(9). setuid, setgid - set user and group ID

(10). stime - set time

(11). sync - update super-block

(12). times - get process times

(13). execl, execv, execle, execve, execlp, exec, exece, environ - execute a file

(14). fork - spawn new process

(15). lock - lock a process in primary memory

(16). utime - set file times

(17). mknod - make directory or a special file

(18). mount, umount - mount or remove file system

(19). time, ftime - get data and time

(20). wait - wait process to terminate

(21). chroot - set the root directory

(22). chown - change owner and group of a file

(23). pipe - create an interprocess channel

(24). ioctl, stty, gtty - control device

3.4.2. Interrupts, Traps and Signals

Before discussing KERN vectors, some background information about interrupts, traps and signals which exist in this system is appropriate. Interrupts are changes in the flow of control caused not by the running program but by something else, usually related to the I/O, such as a keyboard interrupt generated from PDP-11/34. A trap is a kind of automatic procedure call initiated by some condition caused by the program, usually an important but rarely occurring condition, such as system call trap and illegal instruction trap. "Signal" is a UNIX system call. It specifies what action is to be taken when the process

receives notification of a particular 'event' from the kernel or another process. The action to be taken when notification occurs is one of the following:

(1) Ignore the notification

(2) Call a procedure at a specified address

(3) Terminate the process

Those signals generated from other processes or terminal cperations on PDP-11/34 are sent down to the LSI-11 through keyboard interrupts. In the UNIX kernel, signals are scheduled on a last in first out basis. This means when a signal is caught by the UNIX kernel and is being processed, it will be interrupted if another signal occurs. After the latter signal is completely processed, the kernel will continue the processing of the former signal from the point at which it was interrupted. The user program executed on the LSI-11 can catch and send signals just as it does under UNIX environment. The kernel program on the LSI-11 catches not only those signals generated from the PDP-11/34 but also those from LSI-11.

Another feature of the kernel is that it manages a complex environment for manipulating processor priority. The interrupt capability of the LSI-11 Bus allows any I/O device to temporarily interrupt current program execution and divert processor operation to service the requesting device. The processor establishes a vector for each device, which

includes a pair of words. The first word contains the starting address of the interrupt handler, the second word contains a new processor status word (PS). The new PS can raise the interrupt priority level, thereby preventing lower priority interrupts from breaking into the current interrupt service handler. Choices of new PS for each interrupt service handler are listed in Figure 3-3. A more detailed discussion follows.

Running Program	Priority Status (PS)
User program	Ø
System call trap handler	Ø
Clock interrupt handler	7
Keyboard interrupt handler	7
Trap errors handlers	7

Figure 3-3.

1. System Call trap -- priority set to Ø

While executing the system call trap handler, the kernel needs to communicate with the PDP-11/34 about I/O system calls or file system calls. Therefore, the system call trap handler must have the lowest priority to allow keyboard and printer interrupts from the PDP-11/34.

2. Clock interrupt -- priority set to 7

The highest priority(7), disabling all other interrupts, is used in the clock interrupt handler. It is set in order to prohibit keyboard interrupts from occurring during processing the clock interrupt routine. Otherwise the signal queue may be interfered with and become out of order.

This routine can generate an alarm signal if the alarm system call had been called previously. If the clock interrupt occurs while in system mode, especially when a communication system call is under execution, any resulting alarm signal will be delayed until the current system call is completed. If it occurs in user mode, alarm signals will be executed immediately.

3. Keyboard interrupt -- priority set to 7

Data that comes from PDP-11/34 through the console terminal input-keyboard also has the highest priority of interrupt. There are two kinds of data, one is the data for system call communication, another is the information of signals generated from PDP-11/34. The keyboard interrupt handler is capable of distinguishing between these two kinds of data. If the signal information comes in during the sys mode (system call communication), it will be saved into a signal queue. After the system call is completed the kernel will pick up the signal from the signal queue and execute it. If the signal information comes in during user

mode, the kernel will execute the signal action immediately. During the execution of the signal routine, the priority is set to Ø.

The priority of keyboard interrupt can not be lower than that of the clock interrupt. Otherwise, the clock interrupt may occur during keyboard interrupt routine and cause the loss of data. The priority of signals are handled in the similar way as that in UNIX.

3.4.3. KERN Vectors

The kernel controls the PSW which contains a hardware priority with values from Ø to 7. The hardware priority determines which device interrupts are enabled or disabled. When the PSW priority is 7, no device can interrupt. When it is n, only devices with a hardware priority greater than n can interrupt. Whenever an interrupt or trap occurs, the LSI-11 hardware stores the current PS and PC on the current stack and loads a new PS and PC from the vector assigned to the interrupt or trap.

The device interrupt and system trap vector of the kernel program on the LSI-11 is shown in Figure 3-4.

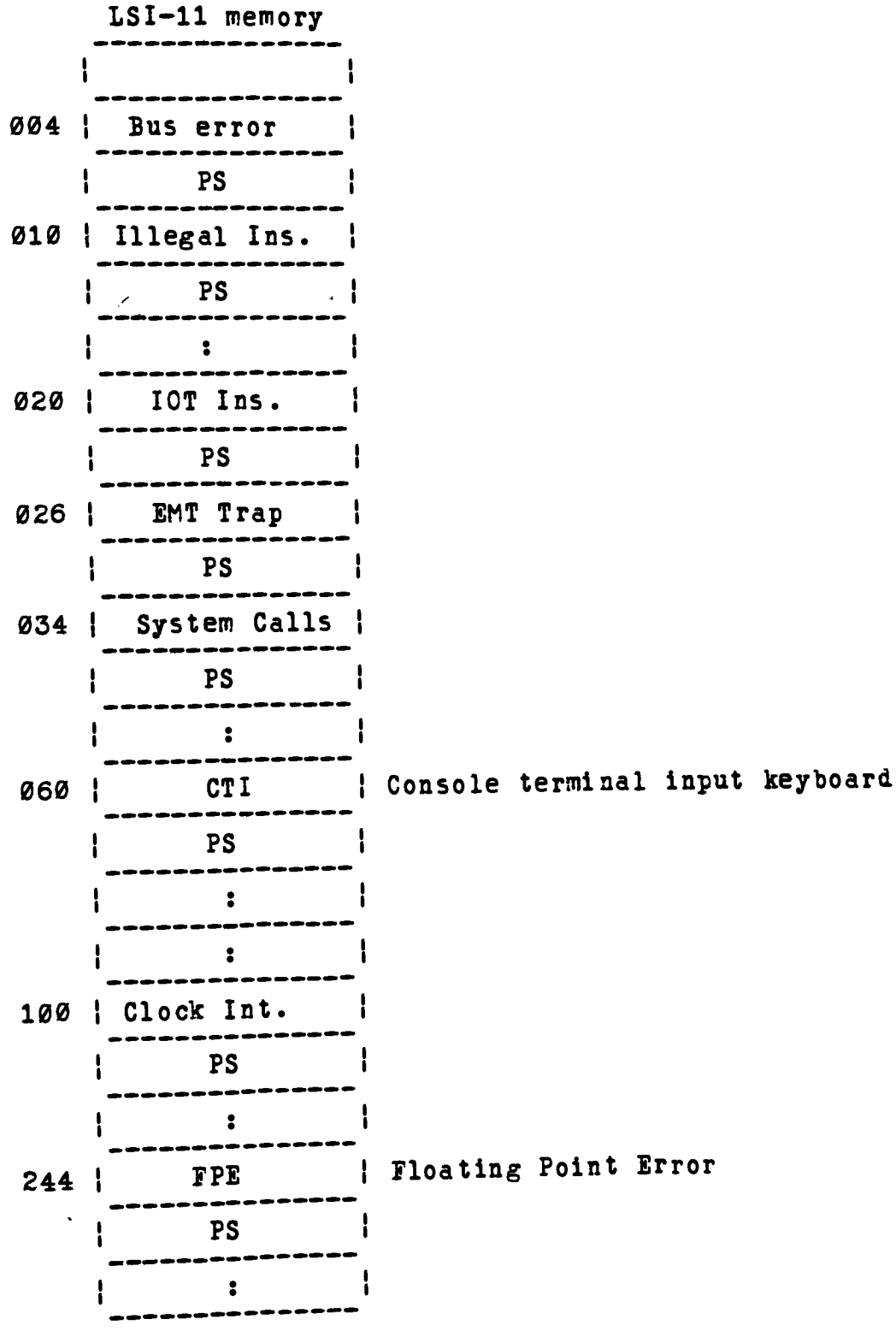


Figure 3-4. The device interrupt and system trap vector

The kernel can be entered due to an action on the part of a user program, a device (console terminal) or the real-time clock interrupt. The following are the three types of entries:

1. Traps - System call, Bus error, illegal and reserved instruction, IOT instruction, EMT instruction.

2. Device Interrupts - Console terminal, input keyboard/read or output printer/punch.

3. Clock Interrupts - External event, line clock interrupt.

As soon as an interrupt occurs, if the priority is higher, automatically jump to the corresponding position and execute the interrupt handler routine.

3.4.4. Implementation of System Calls in KERN

The kernel program catches system call traps within the LSI-11 on behalf of the user program and determines whether to handle them locally or transmit the system call to the PDP-11/34 via the communication network. Those system calls that executed directly on the LSI-11 includes:

(1). indir - indirect system call

(2). alarm - schedule signal after specified time

(3). brk, sbrk, break - change core allocation

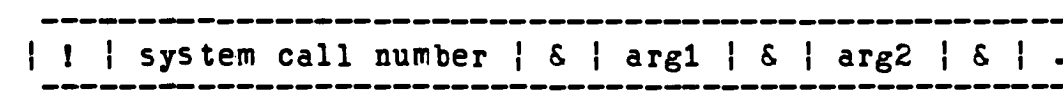
(4). pause - stop until signal They will be discussed more in Chapter 4.

Those system calls that implemented by communicating between PDP-11/34 and LSI-11 are:

(1). access - determine accessibility of file
(2). chdir - change default directory
(3). chmod - change mode of file
(4). close - close a file
(5). creat - create a new file
(6). exit - terminate process
(7). getpid - get process identification
(8). getuid, getgid, gitegid, geteuid - get user and group id
(9). lseek, tell - move read/write pointer
(10). link - link to file
(11). unlink - remove directory entry
(12). open - open for reading or writing
(13). stat, fstat - get file status
(14). read - read from file
(15). write - write on a file
(16). dup, dup2 - duplicate an open file descriptor
(17). umask - set file creation mode mask

A message packet is used for all system call communication. The message packet consists of a control symbol and a system call number followed by its arguments. The arguments are separated from each other by a special symbol. It is shown in Figure 3-5.

Message Packet for system call communication



! Control symbol indicating the beginning of a system call

& Symbol indicating end of argument

Figure 3-5.

The communication message packet is sent to PDP-11/34 in the kernel program whenever a system call is transmitted. Actual size differs for different system calls. The basic element of communication is an 8-bit byte. Messages from the PDP-11/34 to the LSI-11 are variable length strings of bytes containing the result of system calls.

There is one important rule in data communication that needs to be pointed out here. The form of the output data from the LSI-11 sent to the PDP-11/34 is really treated as a terminal by the PDP-11/34. The high order bit of each byte is reserved for special use (e.g., parity). Thus 8-bit binary data must be converted into 7-bit ASCII codes before

transmition in order to avoid confusion regarding the high order bit. Also the system call number and the numerical type of arguments can be identified from the '&' symbols which are placed in between. For the write system call, the output data is transmitted by counting the number of bytes, so there is no '&' symbol at the end.

3.5. LSIRUN Software

3.5.1. The Algorithm of the LSIRUN Program

LSIRUN is a program running on the PDP-11/34 on behalf of the LSI-11. It executes system calls, passes the results back to the LSI-11, and returns control to the LSI-11 user program. During the time that the LSI-11 is executing a program, LSIRUN is roadblocked waiting for an I/O interrupt from the LSI-11. There are two kinds of data which pass between PDP-11/34 and LSI-11. One is the Octal Debugging Data an the other is system call packets. Upon receiving the first character from LSI-11, LSIRUN uses that character to determine the type of data.

The algorithm of the LSIRUN program on the PDP-11/34 follows:

1. Open the LSI-11 communication line, change the terminal interface into cbreak mode, send down the object file which includes the kernel program and the user program loaded together.

2. After the object file is successfully loaded, the user can use ODT commands for debugging until typing a "G" command to load arguments and the environment from LSIRUN.

3. After loading, the user can use the ODT facility; until typing a "P" command to run the program on the LSI-11.

4. Wait for a system call request from the kernel program on LSI-11 or an ODT request.

5. If it is a system call, execute the required system call on the PDP-11/34, and return the results to the LSI-11. If it is an ODT request, change the terminal interface mode to be ready for ODT debugging work. Return to step 4.

When the LSIRUN program starts on the PDP-11/34, the LSI-11 is opened for the user and the KERN file is loaded into the LSI-11. The user's terminal now acts exactly the same as the console for the LSI-11, and ODT is available to the user. After the user program begins, only the halt switch on the LSI-11 or halt instructions will allow the user to return to ODT. At the same time, LSIRUN on the PDP-11/34 recognizes the ODT symbol, which is an at sign (@), and becomes ready to send and receive ODT data for debugging.

3.5.2. Down Loading a Program via LSIRUN

One of the functions of LSIRUN involves down loading the KERN file, together of the user's program, to the LSI-11

memory and setting up the arguments on the stack in the LSI-11. The layout information of the "KERN" object file has four sections: a header, the program text and data, relocation information, and a symbol table. In the header, the size of each section is given in bytes. When the "KERN" object file is loaded into LSI-11 for execution, the text begins at address zero; the header is not loaded. If the magic number in the header is 0407(8) [26], the data segment is immediately contiguous with the text segment. This is the only type of file that can be loaded into LSI-11.

The procedure of loading the whole program into the LSI-11 begins by loading a bootstrap program that enables the LSI-11 to perform input. The LSI-11 requires that the bootstrap program be loaded by switching the BOOT switch on LSI-11. The loading process used by the LSI-11 leaves the CPU at priority 7, which inhibits all interrupts. After loading the boot program, the PC will be set to the first instruction in the boot program, which is now ready to read the " KERN " file. The error message " load error " will appear on the user terminal if any error occurs during loading the object file.

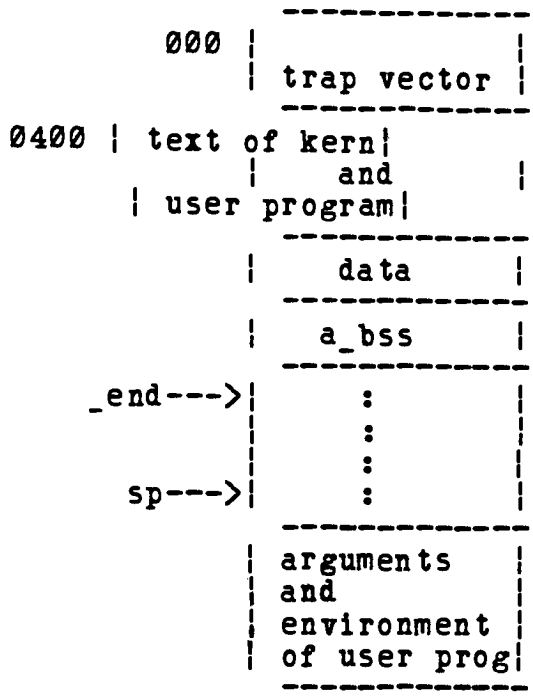
When the LSIRUN begins execution, the following arguments are available:

```
main ( argc, argv, envp )
int argc;
char **argc; **envp;
```

ENVP is a pointer to an array of strings that constitute the environment of the process. When a called file starts execution on the PDP-11, the stack pointer points to a word containing the number of arguments. The structure is as follows:

```
sp-->   nargs
        address of arg0
        .....
        address of argn
           0
        address of env0
        .....
        address of envn
           0
        string of arg0
        .....
        string of argn
           0
        string of env0
        .....
        string of envn
           0
```

The kernel program receives those arguments from LSIRUN and saves them at the bottom of the LSI-11 memory space. They are arranged in the same way as the stack on the PDP-11/34. After the arguments are loaded, the memory on the LSI-11 has the following structure:



3.5.3. Implementation of System Calls in LSIRUN

System calls in LSIRUN are processed by passing the appropriate arguments from the LSI-11 to the PDP-11/34 and invoking the corresponding system call in the PDP-11/34. Some system calls require more elaborate treatment. Their emulation is discussed in more detail here.

To emulate the " signal " system call, a table of signal registers is set in the KERN program, one index for

each possible signal handled by the UNIX system. The same table also exists in the LSIRUN program. This table is consulted before determining the appropriate action to take on the PDP-11/34. The table is initialized to send all caught signals down to the LSI-11, except SIGKILL which cannot be caught. If a signal is set to be ignored by the user in the LSI-11, the corresponding table action on the PDP-11/34 will be also changed to ignore. If such a signal occurs on the PDP-11/34, it will not be sent down to the LSI-11.

The kill system call enables the user program to send signals to other processes running on the PDP-11/34 or to itself. If the user program calls kill, and sends SIGKILL to itself, LSIRUN will also be terminated also. The process running on the LSI-11 is terminated by using the halt switch manually.

The stty and gtty system calls are not really applicable to the LSI-11. If either of them is invoked, it will be sent to the control channel of PDP-11/34.

An exit system call causes LSIRUN to exit and the KERN program to jump to a halt routine for termination.

The most time-consuming system calls to be emulated are read and write. A read system call involves reading from the appropriate file on the PDP-11/34 and then transferring data into the user's buffer on the LSI-11. When a caught signal occurs during this process, the read call terminates

prematurely and causes error.

In order to prevent this difficulty, a control length of buffer size is set while sending data to the LSI-11. The control length is determined on the basis that divisibility operation, e.g. a signal, still can be caught after transmitting such a short length of data.

The write system call involves receiving the transferred data from the user's buffer on the LSI-11 and writing it out to the appropriate file. Since a pointer is used in kernel to monitor the user's buffer, the problem encountered above does not happen here.

The fork, wait, pipe and some other system calls that are not implemented in this system will be trapped if they are invoked by the user program on the LSI-11. The LSIRUN program receives those system calls and prints out a message indicating that those system calls have not been implemented.

4. The User Manual

4.1. Running Programs on the LSI-11 Microcomputers

The LSIRUN program is invoked with the following command:

```
LSIRUN  filename
```

The filename is any name of an object file which is the combination of the kernel program and the user program. If the desired LSI-11 is busy or not available, a message is printed at the terminal. In this case, you must wait until a LSI-11 is available. When one LSI-11 is opened, an identification message is printed to indicate which LSI-11 it is. In the mean time, the LSIRUN program is ready to load the boot program into the LSI-11. User should follow the instructions which are printed out at the terminal and do the following:

(1) Move the ENABLE/HALT switch on the opened LSI-11 to HALT, press the BOOT button, move the ENABLE/HALT switch back to ENABLE.

(2) Move the LTC switch to the OFF position.

(3) Hit the "c" character.

The LSIRUN program now looks for a file whose name matches the "filename" following the LSIRUN. If the file is not found, an error message is printed. If the file is found, LSIRUN begins sending it to the LSI-11.

After the " KERN " file has been loaded, the terminal interface is in " cbreak " mode. An " at " sign (@) is printed on the user's terminal. The LSI-11 console emulator is now in control. Every character that the user types is sent to the LSI-11, and every character the LSI-11 prints out appears on user's terminal. From the time KERN is loaded, the erase and kill characters, rubout, control-s and control-q do not work like they normally do. Instead, they are singly sent to the LSI-11 when the user types them. Console emulator commands may be used at this point to modify the program, or debugging instructions may be inserted or the 'G' or 'P' command may be used to begin program execution.

LSIRUN now is waiting for the user to type in a 'G' command which invokes a procedure to send the argiuments to the LSI-11. After the arguments have been loaded, the user terminal displays an at sign (@). At this time, the user can use ODT or resume execution of his program by typing the " P " command.

The loading procedure used by LSIRUN leaves the CPU at priority 7 which inhibits all interrupts. Before running

the user program, the LTC switch must be moved to the ENABLE position to enable clock interrupts.

LSIRUN program commands are distinguished from data in the ODT program by the tilde (~). The command ~e causes the LSIRUN program to unhook the user's terminal from the LSI-11 and return back to UNIX. No carriage return is needed following typeing ~e.

4.2. System Call Description

(1) ACCESS - determine accessibility of file

```
SYNOPSIS
  access ( name, mode )
  char   *name;

ASSEMBLER
  ( access = 33. )
  sys access; name; mode
```

The sysaccess.c routine on the KERN program sends out the arguments of access system call to LSIRUN on the PDP-11/34 using the format as follows:

```
!access=33 & name & mode &
```

The systaccess.c routine on the LSIRUN program receives those arguments and executes the access system call under UNIX. Zero is returned from successful tests; -1 is returned when an error occurs and the error code is in an error number. The return value and error number are sent back to the LSI-11 and received by the waitback.s routine on the KERN program. If the return value is -1, then the error number is set into r0; otherwise, the return value is set into r0.

(2) ALARM - schedule signal after specified time

```
SYNOPSIS
  alarm ( second );
  unsigned seconds;

ASSEMBLER
  ( alarm = 27. )
  ( seconds in rØ )
  sys alarm
  ( previous amout in rØ )
```

The LSI-11 provides a real-time clock input which generates 6Ø interrupts per sencon. The alarm system call routine in the kernel program is called sysalarm.c and is responsible for setting the system global variable alarmclock to seconds which is given by the arguments of the system call. The alarm clock will be multiplied by 6Ø and decreased by 1 whenever the clock interrupt occurs. Successive calls reset the alarm clock. If the argument is Ø, any alarm request is cancelled. The return value is the amount of time previously remaining in the alarmclock.

(3) BRK,SBRK, BREAK - change core allocation

```
SYNOPSIS
  char  *brk ( addr )
  char  *sbrk ( incr )

ASSEMBLER
  ( break = 17. )
  sys bread; addr
```

The break system call is called from the C library routine brk (addr) and sbrk (increment). Brk sets the system's idea of the lowest location not used by the program to addr. Sbrk increases incr bytes which are added to the program's data space and returns the pointer to the start of the new area.

The system call routine in the kernel program called syssbreak.s, is responsible for checking the new address which must be higher than the current highest location and lower than the current stack pointer. Zero is returned if the break could be set; -1 is returned if the program requests for more memory than the system storage limit. In the second case,the kernel program will set the condition code and set the error value to be 12 which indicates that a program asks for more space than the system can supply.

(4) CHDIR - change default directory

```
SYNOPSIS
  chadir ( dirname )
  char   *dirname;

ASSEMBLER
  ( chdir = 12. )
  sys chdir; dirname
```

The syschdir.c routine in the KERN program sends out the arguments of chdir system call to LSIRUN in the PDP-11/34 using the format as follows:

```
!chdir=12 & dirname &
```

The systchdir.c routine in LSIRUN receives those arguments and executes the chdir system call under UNIX. The return value is zero if the directory is changed; -1 is returned if the given name is not that of a directory or is not searchable. The return value and error number are sent back to the LSI-11 and received by the waitback.s routine in the KERN program. If the return value is -1, an error number is set into r0; otherwise, the return value is set into r0.

(5) CHMOD - change mode of file

```
SYNOPSIS
  chmod ( name, mode )
  char   *name;

ASSEMBLER
  ( chmod = 15. )
  sys chmod; name; mode
```

The syschmod.c routine in the KERN program sends out the arguments of the chmod system call to LSIRUN in the PDP-11/34 using the format as follows:

!chmod=15 & name & mode &

The systchmod.c routine in the LSIRUN program receives those arguments and executes the chmod system call under UNIX. The return value is zero if the mode is changed; -1 is returned if the name cannot be found or if the current user is neither the owner of the file nor the super-user. The return value and error number are sent back to the LSI-11 and received by the waitback.s routine in the KERN program. If the return value is -1, the error number is set into rØ; otherwise, the return value is set into rØ.

(6) CLOSE - close a file

```
SYNOPSIS
  close ( fildes )

ASSEMBLER
  ( close = 6. )
  ( file descriptor in rØ )
  sys close
```

The sysclose.c routine in the KERN program sends out the arguments of the close system call to the LSIRUN in the PDP-11/34 as follows:

```
!close=6 & fildes &
```

The systclose.c routine in the LSIRUN receives those arguments and executes close system call under UNIX. The return value is zero if a file is closed; -1 is returned for an unknown file descriptor. The waitback.s routine in the KERN program receives the return value and error number from the LSIRUN. If -1 is returned, the error number is set into rØ; otherwise, the return value is set into rØ.

(7) CREAT - create a new file

```
SYNOPSIS
  create ( name, mode )
  char    *name;

ASSEMBLER
  ( create = 8. )
  sys creat; name; mode
  ( file descriptor in r0 )
```

The syscreat.c routine in the KERN program sends out the arguments of the creat system call to the LSIRUN in the PDP-11/34 as follows:

```
!creat=8 & name & mode &
```

The systcreat.c routine in the LSIRUN program receives those arguments and executes creat system call under UNIX. The file descriptor is returned from UNIX if the system call has been executed successfully; -1 is returned if error occurs. The return value and error number are sent back to the waitback.s routine on the LSI-11. If the return value is -1, the error number is set into r0; otherwise, the return value is set into r0.

(8) DUP, DUP2 - duplicate an open file descriptor

```
SYNOPSIS
  dup ( fildes )
  int   fildes;

  dup2 ( fildes, fildes2 )
  int    fildes, fildes2;

ASSEMBLER
  ( dup =41. )
  ( file descriptor in r0 )
  ( new file descriptor in r1 )
  sys dup
  ( file descriptor in r0 )
```

The sysdup.c routine in the KERN program must distinguish the dup system call from the dup2 system call. Since the dup2 entry is implemented by adding 0100 to the file descriptor, the KERN program can distinguish dup2 from dup by examining the seventh bit on the dup2 file descriptor. If the 7th bit is 1 then KERN sends out the communication format as:

!dup2=42 & fildes & fildes2 &

Ctherwise, the format is:

!dup=41 & fildes &

The syst2dup.c and systdup.c routines are the corresponding program on the LSIRUN. Both of them receive their arguments and execute dup system call under UNIX. The return value is the file descriptor if the dup system call

is executed successfully; -1 is returned if error occurs. The waitback.s routine receives the return value and error number from LSIRUN. If -1 is returned then the error number is set into rØ; otherwise, the return value is set into rØ.

(9) EXIT - terminate process

```
SYNOPSIS
  exit ( status )
  int   status;

ASSEMBLER
  ( exit = 1. )
  ( status in rØ )
  sys exit
```

The sysexit.c routine in the kernel program sends out the arguments of exit system call as follows:

```
!exit=1 & status &
```

After KERN sends out the arguments of this system call, the kernel program executes a halt instruction which terminates the process on the LSI-11. As soon as the systexit.c routine on the LSIRUN receives the exit system call, the LSIRUN process is terminated by executing the exit system call under UNIX.

(10) GETPID - get process identification

```
SYNOPSIS
  getpid()

ASSEMBLER
  ( getpid = 20. )
  sys getpid
  ( pid in r0 )
```

The sysgetpid.c routine in the KERN program sends out the arguments of this system call as follow:

```
!getpid=20 &
```

The sgetpid.c routine in the LSIRUN program executes the getpid system call under UNIX. The return value is the process ID of the current process. There will be no error, hence no error number can be set. But in order to match the waitback.s routine in the KERN program, both the return value and error number are sent back to the waitback.s routine.

(11) GETUID, GETGID, GETEUID, GETEGID - get user and group identify

```
SYNOPSIS
  getuid()
  geteuid()
  getgid()
  getegid()

ASSEMBLER
  ( getuid = 24. )
  sys getuid
  ( real user ID in rØ, effective user ID in r1 )

  ( getgid = 47. )
  sys getgid
  ( real group ID in rØ, effective group ID in r1 )
```

The sysgetuid and sysgetgid routines in the KERN program send out the following arguments:

```
!getuid=24 &
!getgid=47 &
```

These two routines cannot identify the effective ID calls from the real ID calls. The sgetuid and sgetgid routines in the LSIRUN execute the effective ID together. The return data include the real user ID and the effective user ID for sgetuid routine. They also include group ID and the effective group ID for sgetgid routine. In order to match the waitback.s routine in the KERN program, the getuid and getgid routines still send back an error number which is gnored in this system call.

(12) KILL - send signal to a process

```
SYNOPSIS
  kill ( pid, sig )

ASSEMBLER
  ( kill = 37. )
  ( process number in rØ )
  sys kill; sig
```

The signal "sig" is sent out to the process specified by the process number "pid". The sending and receiving processes must have the same effective user ID.

The user program may send signal to itself or other processes which run on the PDP-11/34. In both situations the KERN program must send the kill system call arguments to the LSIRUN program. The communication format is:

!kill=37 & pid & number &

The user program on the LSI-11 has the same pid as the LSIRUN program on the PDP-11/34. Zero is returned if the signal is sent to the process; -1 is returned if the process does not have the same effective user ID and the user is not super-user, or if the process does not exist. The waitback.s routine receives the return value and error number from LSIRUN program. If -1 is returned the error number is set into rØ; otherwise, the return value is set into rØ.

(13) LINK - link to a file

```
SYNOPSIS
  link ( name1, name2 )
  char   *name1, *name2;

ASSEMBLER
  ( link = 9. )
  sys link; name1; name2
```

The syslink.c routine in the KERN program sends out the arguments of the link system call as follows:

```
!link=9 & name1 & name2 &
```

The systlink.c routine in the LSIRUN program receives those arguments and executes the link system call under UNIX. The return value is zero when a link is made; -1 is returned when an error occurs. The waitback.s routine in the KERN program receives the return value and error number from LSIRUN. If the return value is -1 then the error number is set into r0; otherwise, the return value is set into r0.

(14) INDIR - indirect system call

```
ASSEMBLER
  ( indir = 0 )
  sys indir; call
```

The system call at the location " call " is executed. This system call routine in the kernel program is called sysinidr.s and is responsible for getting the real system call number and saving the pointer which points to the arguments of the system call. If the instruction at the indirect location is not a system call, the kernel sets the condition code to be 1 and returns an error code EINVAL, which is 22, to indicate an invalid argument.

(15) LSEEK, TELL - move read/write pointer

```
SYNPOSIS
  long lseek ( fildes, offset, whence )
  long offset;
  long tell ( fildes )

ASSEMBLER
  ( lseek = 19. )
  ( file descriptor in rØ )
  sys lseek; offset 1; offset 2; whence
```

The syslseek routine in the KERN program sends out the arguments as follows:

```
!lseek=19 & fildes & high offset & low offset& whence&
```

The systlseek in the LSIRUN program receives those arguments and combines the high and low offset values to a " long " type value. The return value of the lseek system call under UNIX is a long integer which is the resulting pointer location. -1 is returned if any error occurs. The waitget.s routine in the KERN program is respnsible for receiving a long-type returned value and the error number. If -1 is return the error number is set into rØ; otherwise, a long integer is saved into rØ and r1.

(16) OPEN - open for reading or writing

```
SYNOPSIS
  open ( name, mode )
  chat  *name;

ASSEMBLER
  ( open = 5. )
  sys open; name; mode
  ( file descriptor in rØ )
```

The sysopen.c routine in the KERN program sends out the arguments as follows:

```
!open=5 & name & mode &
```

The systopen.c routine in the LSIRUN program receives those arguments and executes open system call under UNIX. The return value is a file descriptor if the call executed successfully; -1 is returned if the file dose not exist or is unreadable, or if too many files are open. The waitback.s routine receives the return value and the error number from LSIRUN. If -1 is returned then the error number is set into rØ; otherwise, the return value is set into rØ.

(17) PAUSE - stop until signal

```
SYNOPSIS
  pause()

ASSEMBLER
  ( pause = 29. )
  sys pause
```

This system call routine in the kernel program is called syspause.c. It is used to give up control while waiting for a signal from kill or alarm. This routine is responsible for executing a while loop until the signal occurs.

(18) SIGNAL - ignore or catch signal

```
SYNOPSIS
  #include <signal.h>

  (*signal (sig, func) ) ()
  (*func) ();

ASSEMBLER
  ( signal = 48. )
  sys signal; sig; label
```

Some signals can be originated on the LSI-11 and must be handled on the LSI-11 by the KERN itself. They are SIGILL, SIGTRAP, SIGFPE, SIGBUS, SIGALRM and SIGEMT. Since the kernel program is designed not to generate these signals, they can only be generated form the user program. If they do occur during the kernel program executing, a halt instruction will be executed immediately for system error debugging.

In order for the user program to execute the signal system call, the kernel program in the LSI-11 needs to create a signal table which mimics that on the UNIX operating system. Each signal is initialized by action SIG_DFL which causes a termination of the current process. The signal system call called by the user program allows those signals either to be ignored or to cause an interrupt to a specified location. The syssignal.c routine in the kernel program is responsible for changing the label action on the signal table according to the arguments of the signal system

call. Once the signal occurs, the program starts to execute the label action. The arguments of the signal system call also need to be sent to the PDP-11/34.

Signals also can be generated by other events, such as hitting the terminal break and the request of another program. Those signals are generated on the PDP-11/34, and the LSIRUN program catches them and sends the signal information down to the kernel program through the keyboard interrupt.

The communication type of the signal system call is:

!signal=48 & signal number & signal label &

If the signal label sent from LSI-11 is SIG_IGN then it is executed under UNIX which sets the signal action in the LSIRUN program to SIG_IGN. Once the signal occurs on the PDP-11/34, it should be ignored and need not be sent down to the LSI-11. If the signal is generated from the LSI-11, the kernel program also ignores the signal.

(19) READ - read from file

```
SYNOPSIS
  read ( fildes, buffer, nbytes )
  char   *buffer;

ASSEMBLER
  ( read = 3. )
  ( file descriptor in rØ )
  sys read; buffer; nbytes
  ( byte count in rØ )
```

The sysread.c routine in the kernel program sends out the arguments of the read system call to teh LSIRUN program in teh PDP-11/34. The fromat is:

!read=3 & fildes & nbytes &

In UNIX the return value of read system call is a byte count, which is the number of bytes actually transferred. A return value of zero bytes implies the end of file, and -1 indicates an error of some sort. The return value and the error number are sent from LSIRUN first and received by the waitread.s routine. If an error occurs on this system call then the error number is set into rØ. Otherwise, the LSIRUN sends the read data down to the LSI-11 through keyboard interrupt. The waitread.s routine picks up those data and transfers them into the user's buffer.

(20) STAT, FSTAT - get file status

```
SYNOPSIS
  #include   <sys/types.h>
  #include   <sys/stat.h>

  stat ( name, buf )
  char  *name;
  struct   stat   *buf;

  fstat ( fildes, buf )
  struct   stat   *buf;

ASSEMBLER
  ( stat = 18. )
  sys stat; name; buf

  ( fstat = 28. )
  sys fstat; buf
```

The sysstat.c and sysfstat.c routines in the KERN program send out the format of arguments of the stat and fstat system calls as follows:

```
!stat=18 & name &

!fstat=28 & fildes &
```

The syststat and systfstat routines in the LSIRUN program receive those arguments and execute the same system call under UNIX. The return value is zero if a status is available; -1 indicates that the file cannot be found. The waitstat.s routine in the KERN program receives the return value and error number from LSIRUN. If the system call is executed successfully then the length of the status structure is received first, followed by the status data. The

waitstat.s routine is responsible for receiving the status data and saving them into the buffer address the user declared. If the return value is -1, the error number is set into rØ; otherwise the return value is set into rØ.

(21) UNLINK - remove directory entry

```
SYNOPSIS
  unlink ( name )
  char  *name;

ASSEMBLER
  ( unlink = 1Ø. )
  sys unlink; name
```

The sysunlink routine in the KERN program sends out the arguments of the unlink system call as follows:

```
!unlink=1Ø & name &
```

The systunlink rouitne in the LSIRUN program receives those arguments and executes the unlink system call under UNIX. The return value is zero if this system call is successfully executed; -1 indicates that an error occurs. The waitback.s routine in the KERN receives the return value and error number which are sent from the LSIRUN. If the return value is -1 then the error number is set into rØ; otherwise, the return value is set into rØ.

(22) UMASK - set file creation mode mask

```
SYNOPSIS
  umask ( complmode )

ASSEMBLER
  ( umask = 60. )
  sys umask; complmode
```

The sysumask routine on the KERN program sends out the arguments of the umask system call as follows:

```
!umask=60 & complmode &
```

The systumask.c routine on the LSIRUN program receives those arguments and executes the umask system call under UNIX. The previous value of the mask is returned by the call under UNIX. The return value and error number are returned to the LSI-11 and received by the waitback.s routine on the KERN program.

5. Summary

Without writing an operating system or other supporting software on the LSI-11, the user program can now get the same result as if it were executed directly on the PDP-11/34. This is done by extending the UNIX operating system to the LSI-11. Two programs, LSIRUN on the PDP-11/34 and KERN on the LSI-11, have been implemented to achieve a satellite distributed system. The advantages of this system are:

(1) to save CPU time of the central processor (PDP-11/34).

(2) to increase the utility of the satellite processor (LSI-11).

(3) to support a common pool of peripherals.

(4) to facilitate the use of debugging tools.

(5) to access files in the PDP-11/34's secondary storage.

(6) that any LSI-11 may be located in a remote laboratory location.

For the purposes of this project, the design is such that only one user program can be loaded with the kernel program. A future extension of this project might be to

relocate several user programs under the supervision of one kernel. Without the necessity of loading the kernel for each user program, the efficiency of this system could be improved. It could be achieved by increasing the functions of the dispatcher, the scheduler and the memory manager so that the kernel can handle multiprocess environments.

Appendix (1) Calling Tree Structure for KERN Program

File names ending in .s are assembler language source. Other file names, ending in .c, are C language source.

Other file names, ending in .c, are C language source.

```
start.s
    getw.s
        getc.s
    getc.s
    init.c
        Buserr.s
                enisg.c
                        printout.c
                sigint.c
                        desig.c
                        sysexit.c
        Illins.s
                ensig.c
                sigint.s
        Iotins.s
                ensig.c
                sigint.s
        Emtrap.s
                ensig.c
                sigint.s
        Syscall.c
                sysindir.s
                        syscall.c
                sysexit.c
                        sendcon.c
                        sendcode.c
                             itoa.c
                                 reverse.c
                                     strlen.c
                             printout.c
                             sendfin.c
                        halt.c
                sysread.c
                        sendcon.c
                        sendcode.c
                        waitread.s
                                 putkw.s
                                 putkc.s
```

```
 syswrite.c
         sendcon.c
         sendcode.c
         ptstr.c
         waitback.s
 sysopen.c
         sendcon.c
         sendcode.c
         printout.c
         sendfin.c
         waitback.s
sysclose.c
         sendcon.c
         sendcode.c
         waitback.s

syscreat.c
         sendcon.c
         sendcode.c
         printout.c
         sendfin.c
         waitback.s
syslink.c
         sendcon.c
         sendcode.c
         printout.c
         sendfin.c
         waitback.s
sysunlink.c
         sendcon.c
         sendcode.c
         printout.c
         sendfin.c
         waitback.s
schdir.c
         sendcon.c
         sendcode.c
         printout.c
         sendfin.c
         waitback.s
schmod.c
         sendcon.c
         sendcode.c
         printout.c
         sendfin.c
         waitback.s
sysalarm.c
sysbreak.s
sysstat.c
         sendcon.c
         sendcode.c
         printout.c
         sendfin.c
```

```
                waitstat.s
                        putkw.s
                        putkc.s
        syslseek.c
                sendcon.c
                sendcode.c
                waitget.s
        sysgetpid.c
                sendcon.c
                sendcode.c
                waitback.s
        sysgetuid.c
                sendcon.c
                sendcode.c
                waitget.s
        syspause.c
        sysaccess.c
                sendcon.c
                sendcode.c
                printout.c
                sendfin.c
                waitback.s
        syskill.c
                sendcon.c
                sendcode.c
                waitback.s
        sysdup.c
                sendcon.c
                sendcode.c
                waitget.s
        sysgetgid.c
        sysfstat.c
                sendcon.c
                sendcode.c
                waitstat.s
        syssignal.c
                sendcon.c
                sendcode.c
        sysumask.c
                sendcon.c
                sendcode.c
                waitback.s

        sysfault.c
                sendcon.c
                sendcode.c
Clkint.s
        clkint.c
                ensig.c
Kbint.s
        getc.s
        ensig.c
        sigint.s
```

putq.c

printout.c

Impossible.c

printout.c

Fpe.s

ensig.c

sigint.s

chmod.s

main.c (user program)

Appendix (2) Index of Files in KERN

1. kcomm.h — This file defines all the structures and types needed for communication between the PDP-11/34 and LSI-11.

2. kprocdes.h — This file defines all of the structures and types for dealing with buffers, signal queue and global variables.

3. ksignal.h — This file defines structure and data type of SIGNAL system call.

4. ksyscall.h — This file defines the trap codes used to specify system calls.

5. kchmod.s — This file sets up the stack environment in the same way as the initialization of the UNIX operating system. From here jump to the user's program and change to the lowest priority.

6. kclkint.c — This file checks the global variable alarmclock which is set by the alarm

system call and decreases the variable by 1. If the alarmclock variable becomes zero, then the file will generate the ALARM signal.

7. kdata.c — This file is where all of the shared data area are defined.

8. kdesig.c — This file checks the signal queue if any signal is remaining and then picks up the signal information and calls the sigint.c routine.

9. kensig.c — This file gets signal value from keyboard interrupt buffer and saves it into signal queue.

10. kerror.c — This file contains the trap routines that handles unexpected interrupts. Once it occurs, the error message is printed.

11. kgetq.c — This file picks up one byte from input buffer queue, and passes the value to kuptkw.s file. If the buffer is empty

then wait until the keyboard interrupt occurs and sent the new data to input buffer queue.

12. kgetw.s — This file gets one word from keyboard buffer.

13. kinit.c — This file sets up the interrupt and trap vectors, and initializes the signal tables, global variables, and buffers. It calls the kchmod.s file.

14. kputkw.s — This file calls kgetq.c twice to get one word and pass the word to its calling file.

15. kputq.c — This file is called by the keyboard interrupt routine. It saves the byte into the input buffer queue and wait kgetq.c file to pick up the data.

16. kschdir.c — This fil transmits the chdir system call to the PDP-11/34 as follows: ! chdir & dirname &. It then calls waitback.s rou-

tine to receive the return value and error number.

17. kschmod.c — This file transmits the chmod system call to the PDP-11/34 as follows: ! chmod & name & mode &. It calls waitback.s routine to receive the return value and error number.

18. ksend.c — This file contains several functions which is responsible for sending out data to the PDP-11/34.

19. ksigint.s — This file acts as signal interrupt handler. It picks up the signal from signal queue and processes the signal action.

20. kstartup.s — This file starts to execute the first instruction of the kernel program. It clears memory for loading the environment arguments and calls the init.c file.

21. ksysaccess.s — This file transmits the access system call to the PDP-11/34 as follows: ! access &

file name & mode &. It calls the waitack.s routine to receive the return value and error number from PDP-11/34.

22. ksysalarm.c — This file handles the alarm system call on the LSI-11 itself. It sets the value of alarmclock and return the value of time previously remaining in the alarmclock.

23. ksyscall.c — This file determines which system call is desired and invoks the desired routine.

24. ksysbreak.s — This file handles the break system call on the LSI-11. It checks the memory allocation whether is within correct range or not.

25. ksysclose.c — This file transmits the close system all to the PDP-11/34 as follows: ! close & fildes &. It then calls waitback.s routine to receive the return value and error number.

26. ksyscreat.c — This file transmits the creat system call to the PSP-11/34 as follows: ! creat & name & mode &. It then calls waitback.s routine to receive the return value and error number.

27. ksysdup.c — This file transmits the dup system call to the PDP-11/34 as follows: ! dup & fildes &. It then calls waitback.s routine to receive the return value and error number.

28. ksysexit.c — This file transmits the exit system call to the PDP-11/34 as follows: ! exit & status &. It then executes the halt instruction on the LSI-11.

29. ksysfault.c — This file sends out information of those system calls which are not implemented.

30. ksysfstat.c — This file transmits the fstat system call to the PDP-11/34 as follows: ! fstat & fildes &. It then calls waitstat.s routine to receive the return value and error number. It no error occurs then read in the data and save it into user's buffer.

31. ksysgitgid.c This file transmits the getgid system call to the PDP-11/34 as follows: ! getgid &. It then calls waitget.s routine to receive the return value and error number.

32. ksysgetpid.c This file transmits the getpid system call to the PDP-11/34 as follows: ! getpid &. It then calls waitback.s routine to receive the return value and error number.

33. ksysgetuid.c This file transmits the getuid system call to the PDP-11/34 as follows: ! getuid &. It then calls waitget.s routine to receive the return value and error number.

34. ksysindir.s This file executes the indir system call directly on the LSI-11. It gets the system call number and saves the address of arguments into a global pointer.

35. ksyskill.c This file transmits the kill system to the PDP-11/34 as follows: ! kill & pid & signal number &. It then calls waitback.s routine to receive the return value and error number from PDP-11/34.

36. ksyslink.c — This file transmits the link system call to the PDP-11/34 as follows: ! link & name1 & name2 &. It then calls waitback.s routine to receive the return value and error number.

37. ksyslseek.c — This file transmits the lseek system call to the PDP-11/34 as follows: ! lseek & fildes & high offset & low offset & whence &. It then calls waitback.s routine to receive the return value and error number from the PDP-11/34.

38. ksysopen.c — This file transmits the open system call to the PDP-11/34 as follows: ! open & name & mode &. It then calls waitback.s routine to receive the return value and error number.

39. ksyspause.c — This file handles the pause system call on the LSI-11 itself. It executes a while loop until signal occurs.

40. ksysread.c — This file transmits the read system call to the PDP-11/34 as follows: ! read &

fildes & bytes &. It then calls waitback.s routine to receive the return value and error number. If no error occurs it reads and saves the data into the user's buffer.

41. ksyssignal.c This file transmits the signal system call to the PDP-11/34 as follows: ! signal & sig & label &. It then returns the old label which in the signal table.

42. ksysstat.c This fil transmits the stat system call to the PDP-11/34 as follows: ! stat & fildes &. It then calls a waitstat.s routine to receive the return value and error number from the PDP-11/34. If no error occurs then reads in the data and saves it into the user's buffer.

43. ksysumask.s This file transmits the umask system call to the PDP-11/34 as follows: ! umask & complmode &. it then calls waitback.s routine to receive the return value and error number.

44. ksysunlink.c This file transmits the unlink system call to the PDP-11/34 as follows: ! unlink & name &. It then calls waitback.s routine to receive the return value and error number.

45. ksyswrite.c This file transmits the write system call to the PDP-11/34 as follows: ! write & fildes & nbytes & output data. It then calls waitback.s routine to receive the return value and error number.

46. kvector.s This file contains the vector trap and interrupt routines. It includes clock interrupt, trap system calls, keyboard interrupt, bus error and other trap instructions.

Appendix (3) Calling Tree Structure for LSIRUN Program

```
main
        usage_err
                        emsg
                        exit
        get_lsi
                        makename
                        open_lsi
                        emsg
        quit
                        kill
                        set_tty
                        exit
        open
        set_tty
        identify
                        sprintf
                        write
                        strlen
        init_sig
                        sig_lsi
        get_fork
                        fork
                        sprintf
                        execl
                        exit
                        emsg
        load
                        index
                        emsg
                        getch
                        open
                        strcat
                        badfile
                                emsg
                        fstat
                        read
                        loadunix
                                loadboot
                                        write
                                        sleep
                                newcksum
                                sendword
                                read
                                emsg
                                sendbuf
                                        write
                                sendcksum
                                        sendword
```

```
                write
loadarg
        lodt_loop
                getch
                write
                finish
        write
        sleep
        newcksum
        strlen
        reverse
                strlen
        sendword
        sendarg
                write
        sendcksum
kill
write
wait
saferd
        setjmp
sys_routine
        systexit
                code
                read
                newcksum
                sendword
                sendata
                write
        systwrite
                code
                read
                write
                sendword
        systopen
                open
                code
                read
                sendword
                write
        systcreat
                read
                code
                create
                sendword
                write
        systlink
                read
                malloc
                strlen
                strcpy
                sendword
                free
```

```
systunlink
        read
        unlink
        sendword
stchdir
        read
        chdir
        sendword
stchmod
        read
        code
        chmod
        sendword
        write
syststat
        read
        stat
        sendword
        sendata
        write
systlseek
        lseek
        code
        sendata
        sendword
sgetpid
        getpid
        sendword
sgetuid
        getuid
        geteuid
        sendword
systfstat
        code
        fstat
        sendword
systaccess
        read
        code
        access
        sendword
        write
systkill
        read
        atoi
        code
        kill
        sendword
        write
systdup
        code
        dup
        sendword
```

```
        write
syst2dup
        code
        dup2
        sendword
        write
sgetgid
        getgid
        getegid
        sendword
systsignal
        lsi
        code
        write
        signal
systumask
        code
        umask
        sendword
        write
systfault
        code
        printf
```

Appendix (4) Index of Files in LSIRUN

1. boot.s — This file begins executing on the LSI-11 and ready to read the KERN file from PDP-11/34.

2. lsirun.h — This file contains definitions and declarations of global variables for the LSIRUN program.

3. lsiodt.h — This file contains the name used to invoked the lsiodt program.

ip "4. lfinish.c" 18 This file is called followed by the user typing in ESCAPE(~) symbol druing the ODT mode. The user may exit from kere by typing 'e' character.

5. lget_code.c — This file receives data from LSI-11 of numerical type. It converts those ASCII code into integer value.

6. lget_fork.c — This file generates a child called LSIBACK which used only during ODT mode. It reads input from the lsi and echoes it on the user's terminal.

7. lgetch.c — This fil reads characters from the user's terminal and echoes the character depending on the value of the argument to it.

8. lget_lsi.c — This file contains get_lsi(), makename() and oper_lsi() functions which try to open the LSI-11 for the user.

9. lidentify.c — This file prints out for user which lsi is using now.

10. linit_sig.c — This file initializes signal function. Once signal occurs it executes sig_lsi routine, but exclude SIGKILL because it can not be caught.

11. lmain.c — This file is opens the LSI-11, changes the tty mode, initializes signal routine and gets into communication with LSI-11 for

the system call information and ODT data.

12. load.c — This file opens the KERN file and checks whether the file is appropriate to be executed. If the file is able to be loaded then called the loading routine otherwise point out the error message.

13. loadarg.c — This fil loads the argv and environment arguments down to the LSI-11.

14. loadboot.c — This file sends down the boot.s program which is executed on the LSI-11. When the boot.s program begins executing, it is ready to send the whole object file down to the LSI-11.

15. loadunix.c — This file calls loadboot.c routine and ready to send the whole object file down to the LSI-11.

16. lodt_loop.c — This file responses for ODT mode. It gets data from user's terminal and send the data down to the LSI-11. If user type in

tidle (~) followed by 'e', then exit from the current process. The ODT loop will finish when user type in 'G' or 'P' command.

17. lquid.c — This file terminates the process run on the PDP-11/34 and return the user's tty back to normal mode.

18. lsend.c — This fle contains the functions which send information down to the LSI-11.

19. lset.c — This file sets up the tty mode and stuff. The set_tty() sets the tty to cbreak mode. The reset_tty() resets the tty to normal mode.

20. lsgetgid.c — This file communicates with the LSI-11 of the getgid system call. It calls the getgid and getegid system calls under UNIX and sends back the gid, egid value and error number to the LSI-11.

21. lsgetpid.c — This file communicates with the LSI-11 of the getpid system call. It calls the getpid system call under UNIX and sends back the return ID to the LSI-11. Error number is sent to match the routine on the LSI-11.

22. lsgetuid.c — This file communicates with the LSI-11 of the getuid system call. It calls the getuid and geteuid system calls under UNIX and sends back the uid , euid value and error number to the LSI-11.

23. lsig_lsi.c — This file contains signal routines which transfer signal number to the LSI-11.

24. lsiodt.c — This file is forked from LSIRUN. It reads input from the LSI-11 and echoes it on the terminal while in ODT mode.

25. lstchdir.c — This file communicates with the LSI-11 of the chdir system call. It receives the arguments and calls the chdir system call under UNIX and sends back the return value and error number to the LSI-11.

26. lstchmod.c This file communicates with the LSI-11 of the chmod system call. It receives the arguments and calls the chmod system call under UNIX and sends back the return value and error number to the LSI-11.

27. lsys_routine.c

This file invokes each system call on his own module, and return back the flag which indicate whether it has been successfully done or not.

28. lsyst2dup.c This file communicates with the LSI-11 of the dup2 system call. It receives the arguments and calls the dup2 system call under UNIX. The result value and error number are returned back to the LSI-11.

29. lsystaccess.c This file communicates with the LSI-11 of the access system call. It receives the arguments and calls the access system call under UNIX. The result value and error number are returned back to the LSI-11.

30. lsystclose.c This file communicates with the LSI-11 of the close system call. It receives the arguments and call the close system call under UNIX. The result value and error number is returned back to the LSI-11.

31. lsystcreat.c This file communicates with the LSI-11 of the creat system call. It receives the arguments and calls the creat system call under UNIX. The result value and error number are returned back to the LSI-11.

32. lsystdup.c This file communicates with the LSI-11 of the dup system call. It receives the arguments and calls the dup system call under UNIX. The result value and error number are returned back to the LSI-11.

33. lsystexit.c This file communicates with the PDP-11/34 of the exit system call. It receives the status value from LSI-11 and executes the exit system call under UNIX.

34. lsystfault.c This file receives those system calls which are not implemented on this project.

It prints out the message and the system call number on the user's terminal.

35. lsystfstat.c This file communicates with the LSI-11 of the fstat system call. It receives the arguments and executes the fstat system call under UNIX. It sends the return value and error number to the LSI-11. If no error occurs the length of status structure and the data are sent to the LSI-11.

36. lsystkill.c This file communicates with the LSI-11 of the kill system call. It receives the arguments and executes the kill system call under UNIX. It sends back the retrun value and error number to the LSI-11.

37. lsystlink.c This file communicates with the LSI-11 of the link system call. It receives the arguments and executes the link system call under UNIX. It sends back the return value and error number to the LSI-11.

38. lsystlseek.c This file communicates with the LSI-11 of the lseek system call. It receives the arguments and executes the lseek system call under UNIX. The result value which is long integer and error number are sent back to the LSI-11.

39. lsystopen.c This file communicates with the LSI-11 of the open system call. It receives the arguments and executes the open system call under UNIX. The result value and error number are sent back to the LSI-11.

40. lsystread.c This file communicates with the LSI-11 of the read system call. It receives the arguments and executes the read system call under UNIX. It sends the retrun value and error number to the LSI-11. If no error occurs the read data is sent to the LSI-11.

41. lsystsignal.c This file communicates with the LSI-11 of the signal system call. It receives the arguments and checks the signal label. If the signal label is SIG_IGN the same sig-

nal call is executed under UNIX. Otherwise, the signal table on the UNIX is reset to sig_int() routine.

42. lsystwrite.c This file communicates with the LSI-11 of the write system call. It receives the arguments an executes the write system call under UNIX. The result and error number are sent back to the LSI-11.

Bibliography

1. Bourne, S. R. "The UNIX Operating System," Addison-Wesley, 1983.

2. Bourne, S. R. "UNIX Time-Sharing System: The UNIX Shell," B. S. T. J., 57(6), pp. 1971-1990.

3. Calingaert, P. "Operating System Elements: A User Perspective," 1982.

4. Christian, K. "The UNIX Operating System," John Wiley and Sons, Inc. New York, N.Y. 1983.

5. Davis, William S. "Operating System A Systematic View," Addison-Wesley, 1977

6. Deitel, H. M. "An Introduction to Operating System," Addison-Wesley, 1981.

7. Digital Equipment Corporation, "PDP-11 Paper Trap Software Programming Handbook," 1973.

8. Digital Equipment Corporation, "PDP 11/05/10/35/40 Processor Handbook"

9. Digital Equipement Corporation, "Microcomputers and Memories," 1981.

10. Enslow, P. H., Jr. "What is a distributed processing system?" Computers, vol 11, pp. 13-21, Jan. 1978.

11. Gill, A. "Machine and Assembly Language Programming of the PDP-11," Prentice-Hall:Englewood Cliffs, New Jersey, 1978.

12. Holt, R. C. "Concurrent Euclid The UNIX* System, and Tunis," Addison-Wesley Publishing Company, 1983.

13. Johnson, S. C., Ritchie, D. M. "UNIX Time-Sharing System: Portability of C Programs and the UNIX System," B. J. T. J., 57(6), pp. 2021-2048.

14. Kernighan, B. W., Ritchie, D. M. "The C Programming Language," Prentice-Hall: Englewood Cliffs, New Jersey, 1978.

15. Lister, A. M. "Fundamentals of Operating System," London:Macmillan, 1975.

16. Lycklama, H., Christensen, C. "UNIX Time-Sharing System: A Minicomputer Satellite Process System," B. S. T. J., 57(6) pp. 2087-2101.

17. Lycklama, H. "UNIX Time-Sharing System: UNIX on a microprocessor," B. S. T. J., 57(6) pp. 2087-2101.

18. Lycklama, H. "UNIX Time-Sharing System: The MERT Operating System," B. S. T. J., 57(6), pp. 2049-2086.

19. Needham, R. M., Herbert, A. J. "The Cambridge Distributed Computing System," Addison-Wesley, 1982.

20. Custerhout, J. K., et al.; "Medusa: An Experiment in Distributed Operating System Structure," Communications of the ACM; Vol. 23, No. 2, Feb. 1980.

21. Ritchie, D. M., Johnson, S. C., Lesk, M. E., Kernighan, B. W. "UNIX Time-Sharing System: The C Programming Language," B. S. T. J., 57(6), pp. 1991-2019.

22. Ritchie, D. M., Thompson K. "The UNIX Time-Sharing System," B. S. T. J., 57(6), pp. 1905-1929.

23. Singer, M. "PDP-11 Assembler Language Programming and Machine Organization," Wiley, 1980.

24. Tanenbaum, A. S. "Computer Networks," Prentice-Hall, 1981.

25. Thomas, R., Yates, J. "A User Guide to the UNIX System," Osborne/McGraw-Hill Berkeley, California, 1982.

26. Thompson, K., Ritchie, D. M. "UNIX Programmer's Manual," Seventh Edition, Bell Laboratories: Murray Hill, New Jersey, 1978.

27. Tsichritzis, D. C. "Operating System," New York, Academic Press 1974.

28. Weitzman, Cay. "Distriubuted Micro/Minicomputer Systems: Structure, Implementation, and Application," Prentice-Hall: Englewood Cliffs, New Jersey, 1980.

29. Wilkes, M. V., Needham, R. M. "The Cambridge Model Distributed System," Cperating System Review; Vol. 14, No. 1, Jan. 1980.

30. Wulf W. A., Hartison, S. P. "Reflections in a pool of Processors-An experience Report on C.mmp/Hydra," Processings of the Naticnal Computer Conference, 1978; AFIPS Press

31. Wulf, W., et al.; "Hydra: The Kernel of a Multiprocessor Operating System," Communications of the ACM; Vol. 17, No. 6, June 74